DOCUMENT RESUME

ED 178 095 IR 007 845

AUTHOR Muiznieks, Viktors
TITLE MSEA--The Department's Microcomputer. The Illinois Series on Educational Applications of Computer. Report No. 27.
INSTITUTION Illinois Univ., Urbana. Dept. of Secondary Education.
PUB DATE Dec 78
NOTE 31p.; For related documents, see IR 007 782, IR 007 796, and ED 142 199

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS Computer Programs; *Computers; *Equipment Utilization; Higher Education
IDENTIFIERS *Microcomputers

ABSTRACT

This report provides a technical description and operating guidelines for the IMSAI 8080 microcomputer in the Department of Secondary Education at the University of Illinois. An overview of the microcomputer highlights the register array, address logic, arithmetic and logical unit, instruction register and control section, and the data bus buffer. The timing control, data bus, and interrupt control of the 8080 CPU are described, exercises to familiarize users are provided, and assembly language is discussed. (Author/RAO)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED178095

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
NATIONAL INSTITUTE OF EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

Number 27

The Illinois Series

on

Educational Applications of Computer

MSEA - The Department's Microcomputer

Viktors Muiznieks

December 1978

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Bruce Hicks

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

Department of Secondary Education
College of Education
University of Illinois
Urbana, Illinois

© Bruce Hicks

IR 007 845

4.21

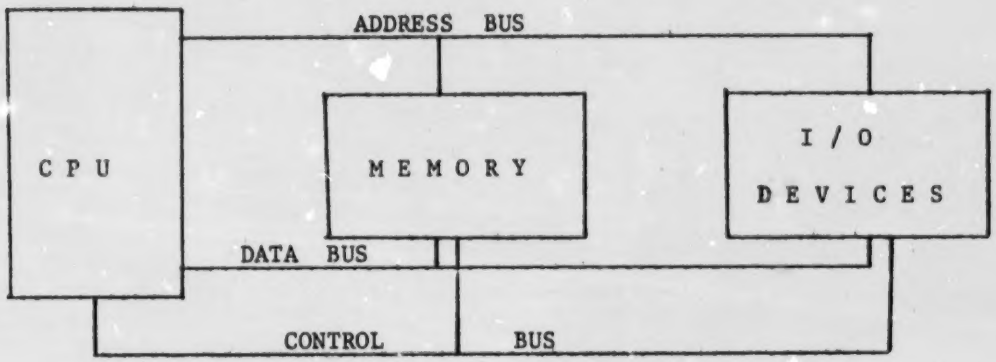
Table of Contents

MSEA -- The Department's Microcomputer

	page
Our Own Micromputer -- An Overview	1
1) The register array and address logic	2
2) The arithmetic and logical unit (ALU)	4
3) The instruction register and control section	4
4) The data bus buffer.	4
A Closer Look at the 8080 CPU	7
1) Timing Control	9
2) Data Bus Definition	10
3) Interrupt Control	10
Functions of the Lights and Switches on the Front Panel	11
Familiarity Exercise	15
Focus on the Assembly Language for Our Microcomputer	17
Other Useful Things to Know about Assembly Language Programming	21
Fundamental Software Concepts	23
References	27

Our Own Microcomputer --An Overview

The basic structure of our system can be visualized as the following:



The I/O (input/output) devices are needed for communication between the internal representation of information which is stored as electronic signals inside the computer and the external representation of that information in a form which is readily understandable to humans (e.g., cards, printed paper, a TV screen, etc.). The various I/O devices available on our system will be described in a later section; they include a card reader, disk drive, cassette recorder, line printer, various terminals and a TV monitor.

The MEMORY is used to store the data, instructions and various intermediate results used by one or more programs. Our system can reference 65,536 (64K, where K represents 1024, or 2^{10}) words of 8-bit RAM memory. The memory is contained on semiconductor chips and is therefore volatile (i.e., information stored there is lost if electric power is turned off).*

Information is placed into the memory either by I/O devices or by the CPU (discussed later). Thus, the total amount of information which our system can

* A similar ROM (non-volatile) memory of 4K words makes it possible to read in the DOS (disk operation system) from diskettes.

access is virtually limitless, since information can be taken from disks, tapes or a terminal keyboard and placed into the memory. After the information has been processed (or a program has been completed), the results (or instructions) can be copied from the memory onto a disk or cassette where they can be stored permanently. In this way, memory space is "freed up" inside the computer for the next user.

The CENTRAL PROCESSING UNIT (CPU) is the most complicated component of the system. It can be viewed as being composed of the following:

1) the register array and address logic,
2) the arithmetic and logical unit (ALU),
3) the instruction register and control section,
4) the data bus buffer.

(1) The register array and address logic

The register array consists of six 8-bit, general-purpose registers, along with a program counter register, a stack pointer register, and two temporary registers (called W and Z). The two temporary registers are used only for internal purposes and cannot be accessed by a programmer when writing a program. Other registers also exist within the CPU, but they logically belong to different parts of the CPU, even though they are located next to each other physically; these registers will be discussed when appropriate.

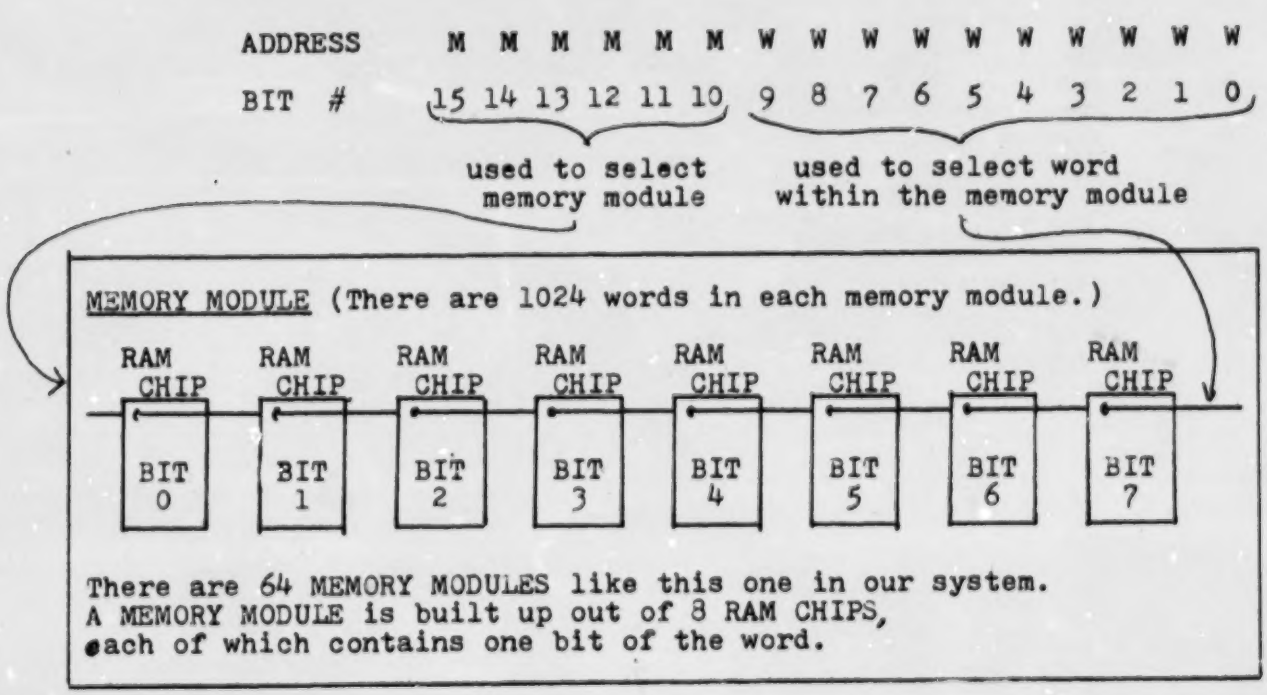
The general purpose registers can be used either individually or in pairs. Register-pairs are frequently used to indicate memory addresses. The registers are paired in the following way: B and C, D and E, and H and L.

The program counter contains the memory address of the program instruction being executed, and it is automatically incremented by 1 after each instruction is fetched. The stack pointer contains the address where the item most recently added to the stack resides. If data is added onto or removed from the stack, the stack pointer is automatically re-adjusted to point to the item which is now at the top of the stack.

The address logic

The address information stored in registers can be interpreted in order to access the corresponding memory locations, and data and instructions can be moved around between the registers and the memory. This is accomplished in a rather complex way, in which each address is interpreted as consisting of memory-module-select bits and word-select bits. The leftmost six bits of an address (the memory-module-select bits) represent the memory module within which the word being addressed is to be found. Since $2^6=64$, 64 different memory modules can exist. The rightmost 10 bits of the address (the word-select bits) are used to select which word within that module is being referenced. Since $2^{10}=1024$, 1024 (or 1K) words can exist within each memory module. We see that 64x1024 (or 64K) different words (with 65,536 unique addresses) can be identified using a sixteen-bit address representation scheme. A diagramillustrating the above discussion is given below:

ADDRESS DECODING


ADDRESS M M M M M M W W W W W W W W W W
BIT # 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
used to select memory module
used to select word within the memory module
MEMORY MODULE (There are 1024 words in each memory module.)
RAM CHIP
BIT 0
BIT 1
BIT 2
BIT 3
BIT 4
BIT 5
BIT 6
BIT 7
There are 64 MEMORY MODULES like this one in our system.
A MEMORY MODULE is built up out of 8 RAM CHIPS,
each of which contains one bit of the word.

(2) The arithmetic and logical unit (ALU)

The ALU contains the specialized circuitry where all the mathematical and logical (AND, OR, NOT) operations take place. Facilities are also provided to shift bits to the right and to the left and to complement the bits in a word (change 0's into 1's and 1's into 0's). The ALU contains an 8-bit register known as the accumulator (register A) where the results of mathematical operations are usually stored first. The ALU also references the W and Z register-pair in performing some of its operations. A 5-bit flag register also exists in the ALU, but flag bits will not be discussed here. (See the section on Fundamental Software Concepts).

(3) The instruction register and control section

The contents of the instruction register direct the activities of the ALU and provide the control signals to the register array and address logic sections. A complex network in the CPU called a control function sequences and directs the processing of the commands stored in the instruction register. Its activities overlap those of other parts of the machine.

What regulates and coordinates all of the activities going on inside the computer are clock signals, which are merely electronic pulses produced at regular time intervals so as to synchronize all of the logic activities within the CPU.

(4) The data bus buffer

The data bus buffer is used so that internal signals can be transmitted within the CPU without accessing the normal data bus (used for transmitting data and addresses between memory, various I/O ports, and the CPU).

The address bus is used to identify a particular memory location or I/O device. The control bus identifies which, if any, of the following activities is in process:

(a) memory read, (b) memory write, (c) I/O read, (d) I/O write, (e) interrupt acknowledge.

The system buses normally cooperate in processing a command in the following way: (a) the control bus identifies the activity which is to be performed, (b) an address is placed on the address bus to identify which particular memory location or I/O device will be involved; (c) the CPU receives from or transmits to the data bus the data with the selected memory location or I/O device and (d) the entire cycle is repeated.

There are ways in which the normal activity of the CPU (i.e., processing of commands in a program) can be altered. (a) interrupt signal may be received. This signal is a request made by a device external to the CPU while a program is being executed. It usually requires some task to be performed. If the CPU is in a state where it will allow interrupts to occur (i.e., interrupts are enabled), then the program is temporarily "suspended" while the CPU takes care of whatever the interrupt wants it to do. The CPU can also be in a state where interrupts are disabled. While in this state, the CPU will ignore an interrupt and continue with what it is doing.

(b) A wait signal tells the CPU that it should wait for a while before going on with what it wants to do because something for which it has asked (maybe something from a slow I/O device) has not yet arrived. As soon as the something arrives, the CPU will be told that it can proceed.

(c) The CPU can also enter a hold state, which is similar to a wait state in that the CPU is told not to do anything for a while. The hold state differs from the wait state in that it is used when an I/O device wants to communicate directly with memory (without going through the CPU); an I/O device may want to transfer directly a large amount of data into memory, for example, and not bother the CPU at all.

In order to protect against the conflict situation where the processer and I/O device might both want to change the contents of the same memory location, the CPU is told to hold-off on what it is doing until the I/O device indicates with a "go-ahead" signal that it has finished what it wanted to do.

The CPU takes several steps to process an instruction. An instruction cycle is defined as the time required to retrieve and execute an instruction. Not all instructions take the same length of time to complete, and not all instructions even have the same length (in terms of the number of words required to specify all of the information needed for the instruction to be executed). (See the section on Fundamental Software Concepts). Some functions require several words to store not only the actual instruction but also all of the addresses and register information required to complete them. The CPU can recognize when an instruction should specify such extra information, and it follows a predefined scheme to look for where that information should be stored (for example, addresses are stored in consecutive memory locations following the instruction itself).

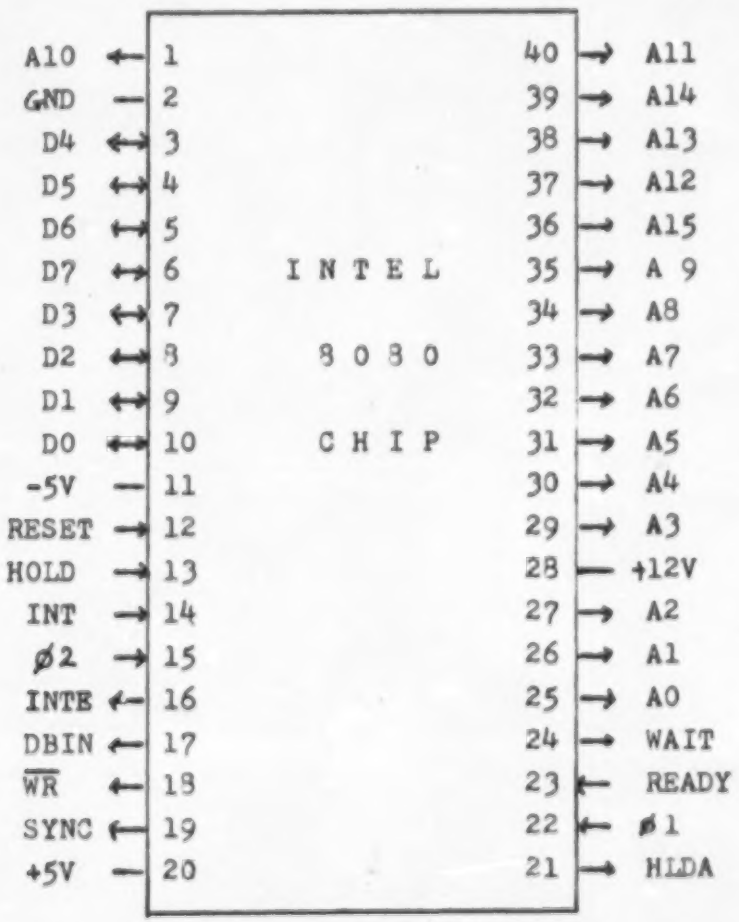
Because instructions involve various operations, the concept of a machine cycle is introduced. A machine cycle is required every time the CPU accesses memory or an I/O port, and instructions are executed as a sequence of one or more machine cycles. Each machine cycle is in turn further broken down into three to five states. A state is the smallest unit of processing activity which can go on in the CPU, and is defined as the time period between two successive clock signals produced by the external clock which synchronises the operation of the entire computer.

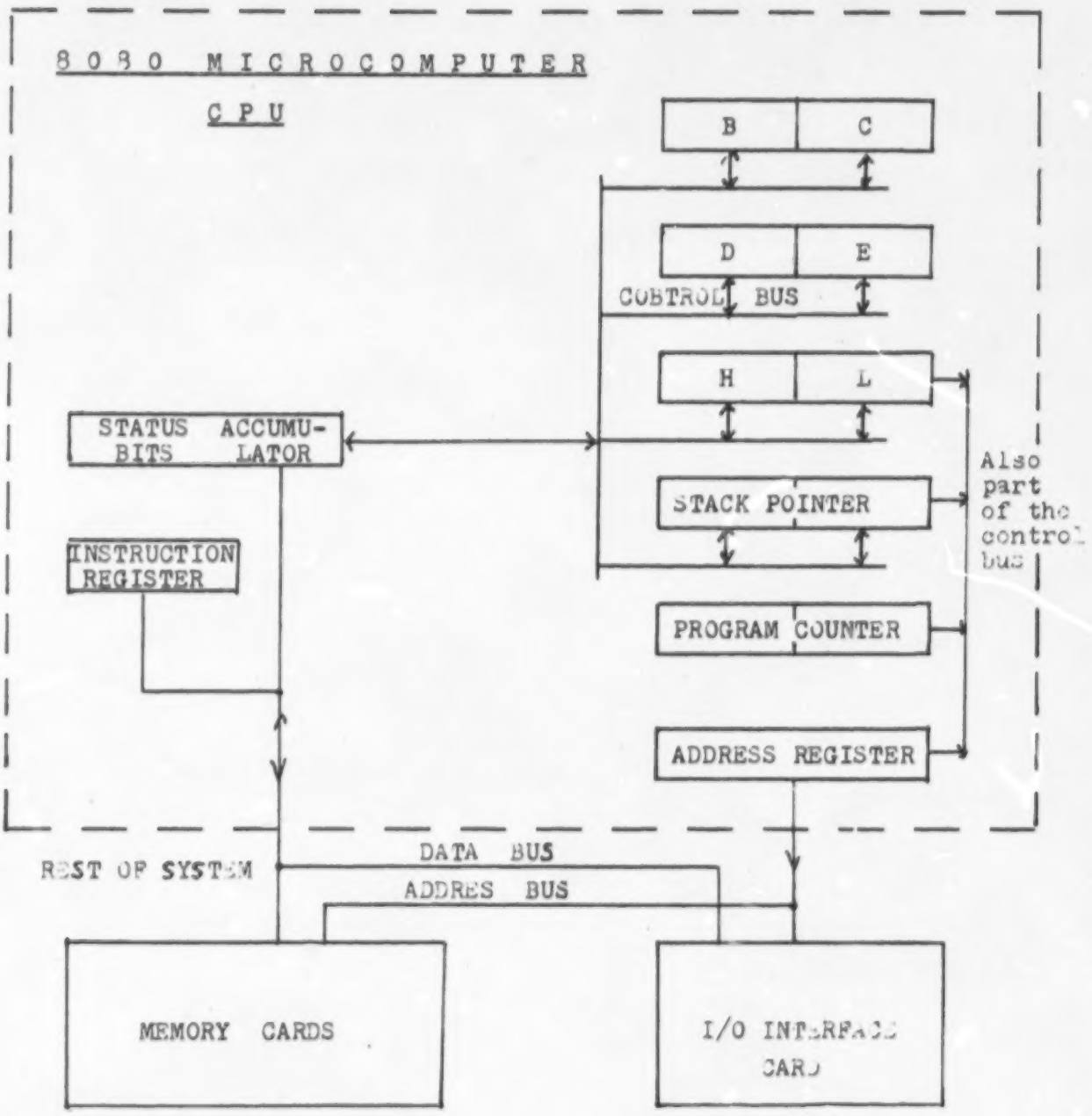
Each clock period marks a state; three to five states constitute a machine cycle, and one to five machine cycles are necessary to perform one instruction cycle. An instruction may require from four to eighteen states for completion; the exact number depends on the particular instruction and on what that instruction does.

An expansion of the simplified model initially presented to describe the various components of the microcomputer is given on the next page. The diagram focuses on the elements of the CPU and indicates how various components are connected to the rest of the system. A rough idea as to how the buses are set up is also presented, although the picture is not entirely complete (particularly in the case of the control bus). The arrows on the bus lines indicate the direction in which the information being transmitted flows.

A Closer Look at the 8080 CPU

The entire CPU of the microcomputer is stored on a single LSI chip, called the Intel M8080A (or just 8080). This chip has 40 pin connections on it which are used to transmit the signals which contain all of the information going into or coming out of the CPU.




8080 MICROCOMPUTER
CPU
B
C
D
E
CONTROL BUS
H
L
STATUS BITS
ACCUMULATOR
STACK POINTER
Also part of the control bus
INSTRUCTION REGISTER
PROGRAM COUNTER
ADDRESS REGISTER
REST OF SYSTEM
DATA BUS
ADDRES BUS
MEMORY CARDS
I/O INTERFACE CARD

Here is an extended discussion of what each of the pins are used for. AØ-A15 define the address bus, which can access up to 64K words of memory, or any of up to 256 input or 256 output devices. AØ is the least significant address bit.

DØ-D7 define the data bus, which is used for communication between the CPU, memory, and the I/O devices for instructions and data. These pins also send signals which define the status word of the CPU during the first clock cycle of each machine cycle. The status work is displayed on the front panel of the microcomputer. When DØ-D7 contain data, DØ is the least significant bit.

The SYNC pin provides a synchronising signal which indicates the beginning of each machine cycle.

The GNP, -5. +5, and +12 V pins are power sources and ground connections which are required for functioning of the CPU. They provide the source of all electric power for the system.

The rest of the pins transmit signals which fall into the following three categories: timing central, data bus definition, and interrupt control signals.

1) Timing control

The READY signal indicates that input data or data from memory is available on the data bus. This signal is used to synchronize the CPU with slower memory devices or I/O devices. If the CPU does not receive a READY signal after sending out an address, it will enter a wait state until the READY signal is received. A WAIT signal acknowledges that the CPU is in a wait state. The READY signal can also be used to control the operation of the SINGLE STEP option on the front panel of the computer (see the section on the Front Panel). The CPU enters a wait state each time the SINGLE STEP switch is hit, and it does not come out of that wait state until the SINGLE STEP switch is hit again. After the switch has been hit the second time, the CPU will acknowledge a ready signal and execute exactly one machine cycle before returning to the wait state.

The HOLD signal requests the CPU to enter the hold state, thus allowing an external device to gain control of the address and data buses as soon as the CPU completes its current machine cycle. The CPU responds to the HOLD request by acknowledging the request on the HLDA pin. The external device knows it can use the data and address buffers as soon as it receives the HLDA (hold acknowledge) signal.

The RESET signal causes the contents of the program counter register to be cleared (in effect, it causes address 0 to be placed into the program counter). The HLDA pin is reset so as to indicate no acknowledged hold state, and the INTE pin is reset to indicate that interrupts are enabled. The flags, accumulater, stack pointer, and other general purpose registers are not zeroed.

Ø1 and Ø2 are two clock signals which aid in synchronising all of the internal operations of the CPU.

2) Data Bus Definition

DBIN (data bus in) is used to indicate to external circuits that the data bus is in the input mode. As soon as the signal goes to a 1, data from an addressed memory location or port is placed onto the data bus.

$\overline{WR}$ (write) is used to control writing onto memory or to an I/O port. The data on the data bus is available when $\overline{WR}$=0.

3) Interrupt Control

INTE (interrupt enable) is used to indicate whether or not the CPU will allow itself to be interrupted. The INT (interrupt request) signal is recognized by the CPU when it finishes processing the current instruction or when it is in a halt state: if the CPU is in the held state or if the INTE signal is such that the CPU will not accept interrupts, then the request will not be honored.

The chart given below can be used as a quick guide to the functions served by each pin and to its type of use as a input pin, an output pin, or both (a bidirectional pin). The direction of information flow is into the CPU for a pin to be of input type and out of the CPU for an output type.

Pin Name	Description	Type
A0-A15	Address lines	Output
D0-D7	Data bus lines	Bidirectional
SYNC	Machine Cycle Synchronizer	Output
DBIN	Data input indicator	Output
READY	Data ready indicator	Input
WAIT	CPU in wait state	Output
$\overline{WR}$	Data output indicator	Output
HOLD	CPU entered in hold state	Input
HLDA	Acknowledge that CPU in hold state	Output
INT	Interrupt request	Input
INTE	Interrupt enable	Output
RESET	Reset parts of the CPU	Input
$\phi 1, \phi 2$	Clock signals	Input
GND, V's	Ground and power signals	Bidirectional

Functions of the Lights and Switches on the Front Panel

Lights and switches can both be thought of as representing 0's and 1's. If a light is turned on or if a switch is in the up position, then the corresponding item has a value of 1; if a light is turned off or a switch is in the down position, the corresponding item has a value of 0. Exceptions occur in the cases where a switch is used to distinguish between two different operations. For example, the switch labelled RUN/STOP, when flipped up, causes the computer to begin executing a program, while flipping the switch down causes the computer to stop the execution of a program. Also, a design error in the IMSAI system causes the output lights to display values that are just the opposite of what they should be; therefore, results displayed on the output lights should be complemented to conform to the convention that a light being on represents a 1 and a light being off represents a 0.

The output lights indicate values output to them by a program that is being executed. They can represent a binary value in the range 0-255. Each light

represents a power of 2, with the highest power (2^7) being represented by the leftmost light, and the lowest power (2^0) being represented by the rightmost light.

There are three sets of switches on the front panel. The two sets of non-control switches (the two leftmost sets) serve different purposes at different times. They can be used in conjunction with each other to specify a memory address, or they can be used separately for inputting data to an executing program or for entering data and instructions into memory locations when writing a program.

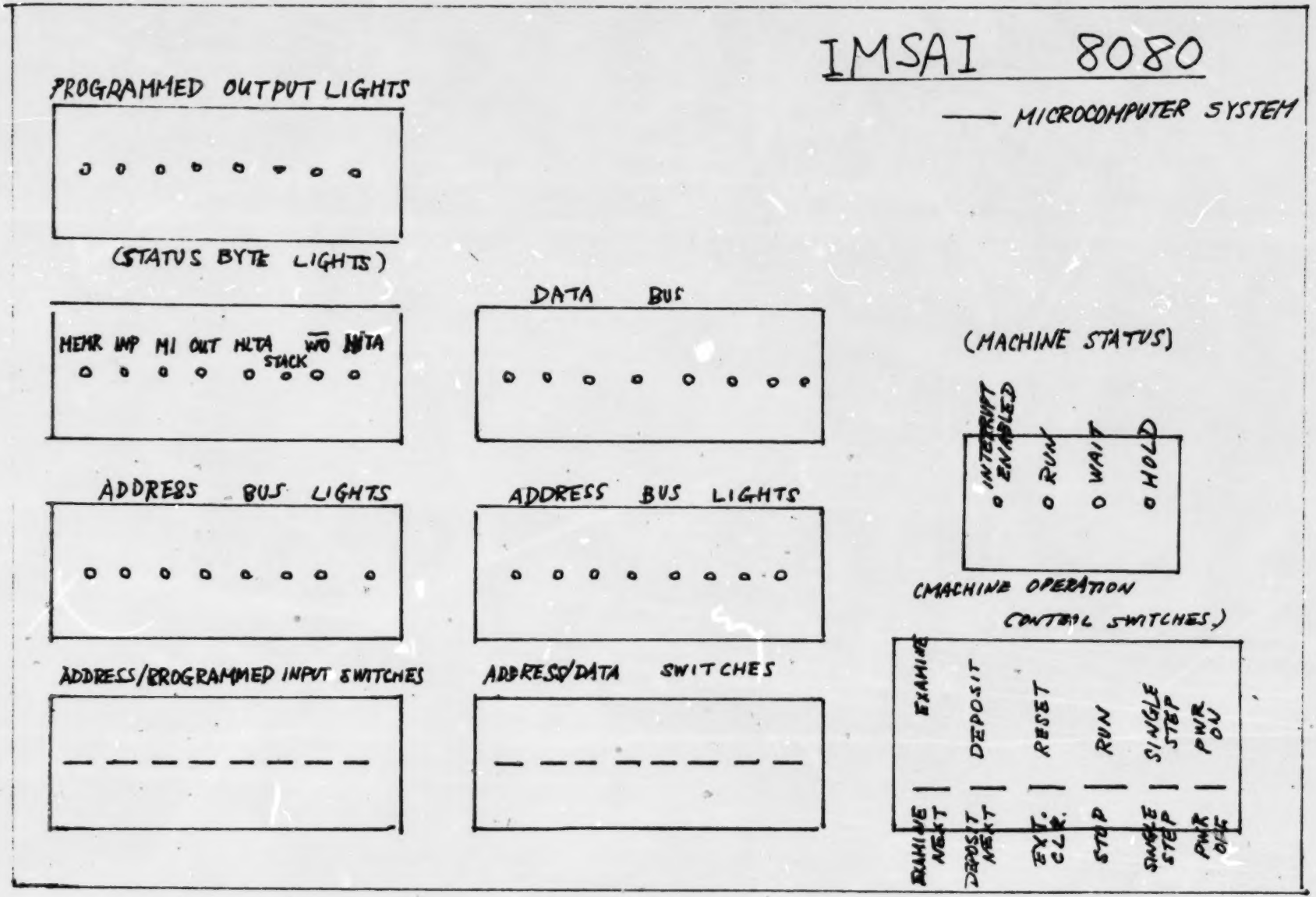
To specify a memory address, all 16 switches must be used (since there are 2^{16} = 65,536 available memory addresses each of which must be uniquely identifiable). Each switch represents a power of 2 (the power is actually shown above the switch on the panel), and the leftmost switch denotes the highest power of 2, while the rightmost switch indicates the lowest power of 2.

To input data into a program that is running, the leftmost eight switches are used: these are capable of representing numbers in the range of 0-255. The powers of 2 which each switch represents are also indicated on the panel; the highest power of 2 is represented by the left most switch.

To enter instructions or data into specific memory locations, the middle set of switches are used. The same convention as above is followed for representing powers of 2.

The address lights indicate the address currently being referenced, be it for examination, reading, or writing purposes. As explained above, 16 such lights are needed, and the powers of 2 which each light represents follows the standard convention.

The data bus lights indicate the value stored at the specific memory address which is displayed on the address lights.



THE FRONT PANEL

The machine status lights indicate what is happening in the CPU. They show when interrupts occur and whether or not the CPU is in a halt, wait, or hold state. If the interrupts enabled light is on, interrupts will be allowed to occur. If a light representing one of the three states is on, it indicates that the CPU is in that state.

The machine control switches are used to control what is being referenced, displayed, or examined. The EXAMINE/EXAMINE NEXT switch is used to examine specific memory locations. When the EXAMINE option is chosen, the address specified on the address switches will be accessed, and the corresponding address will appear on the address lights. The data stored at that address will appear on the data bus lights. If the EXAMINE NEXT option is taken, the next address will be displayed in a similar fashion. The next address is the current address plus 1.

The DEPOSIT/DEPOSIT NEXT switch is used to enter the value specified on the data switches into the address indicated on the address lights. If the DEPOSIT NEXT option is taken, the data present on the data switches is stored in the address which is one greater than that displayed on the address lights.

The RESET/EXT. CLEAR switch repositions the address to location 0 when the reset option is taken. If the EXT. CLEAR option is chosen, all the I/O ports are cleared, and any memory-protected areas are left unprotected.

The RUN/STOP switch is used to tell the computer when to begin and finish executing a program. The address at which programs will usually start is location 0.

The SINGLE STEP/SINGLE STEP switch can be flipped up or down, and this action causes one machine cycle to be executed and the results to be displayed on the various lights.

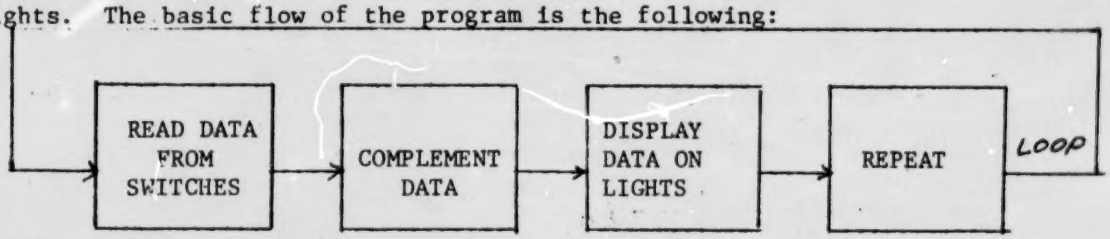
The PWR ON/PWR OFF switch turns power for the machine on or off.

The most complicated lights to understand but the ones which are of the greatest value in debugging a program are the status lights which indicate machine cycle by machine cycle not only what is happening in the CPU but also the various signals being sent between different sections of the system. The following are descriptions of each light;

MEMR*	When lit, indicates that the data bus will be used for memory read data.
INP*	When lit, indicates that the address bus contains the address of an input device and that the input data should be placed on the data bus when DBIN (an internal signal that connects directly to the 8080 CPU) has a 1 on it.
M1	When lit, the address bus contains the location of the instruction being fetched while the data bus contains the instruction.
OUT	When lit, indicates that the address bus contains the address of an output device and that the data bus will contain the output data when $\overline{WR}$ (another internal signal) is a 1.
HLTA	When lit, indicates that the CPU acknowledges a signal for a HALT instruction (an internal signal).
STACK	When lit, indicates that the address bus holds the pushdown stack address from the stack pointer register.
$\overline{WO}$	If the light is not lit, the operation in the current maching cycle will be a WRITE memory or OUTPUT function. Otherwise, a READ memory or INPUT operation will be executed.
INTA*	When lit, indicates acknowledgement of a signal for an INTERRUPT request. The INTA signal will be used to gate a restart instruction onto the data bus when DBIN (the internal signal) becomes a 1.

Familiarity Exercise

The purpose of this exercise is to get you started using the microcomputer and to allow you to become acquainted with the various parts of the front panel. You will be asked to enter a machine language program into the computer by using the input switches and to display a representation of those switches onto the output lights. The basic flow of the program is the following:



*These status lights are used to control the flow of data onto the data bus. When the DBIN (data bus input) signal is a 1, data from an addressed memory location or I/O port can be placed onto the data bus. $\overline{WR}$ is a 0 when the data is on the data bus.

Here is a program listing and related comments: extra information is included which will not be needed at this time, but which may be useful for later reference.

Address	Mach.Lang.	Octal	Hex.	Ass.Lang.	Comments
0	11011011	333	DB	IN	Tell computer to get ready for
1	11111111	377	FF		input coming from switches.
2	00101111	057	2F	CMA	Complement input.
3	11010011	323	D3	OUT	Prepare for outputting
4	11111111	377	FF		output to lights
5	11000011	303	C3	JMP	Loop back to beginning of
6	00000000	000	00		program by executing a jump to
7	00000000	000	00		location 0.

The octal and hexadecimal representation of the machine language instruction are given to provide practice in converting to the different number systems, and the assembly language instruction equivalents are indicated where appropriate.

You can see that the same "port" FF is indicated for both inputting and outputting. This particular port is predefined, and it uses the switches as the input port and the lights as the output port. Notice that the contents of the switches have to be complemented before being displayed to conform to the rules namely that a light being on should indicate a 1, and a light being off should indicate a 0. Also notice that addresses 5, 6 and 7 combine to form one instruction and that it takes two words of memory to indicate a memory location.

Off we go!

1) Turn PWR ON switch to on (UP). If you don't hear some noise, and if the lights do not come on, see if the machine is plugged in.

2) Set address switches to 0 (all leftmost switches and middle switches down) and flip EXAMINE switch up.

3) Enter the machine language instruction which should be at this location into the data switches (the middle set). The instruction is given on the previous page as 11011011. Flip the DEPOSIT switch up. You have now entered your first instruction into the memory, and it is stored in location 0 (or at address 0, if you prefer).

4) Repeat steps 2 and 3 but setting address switches to 1 the first time and to 2 the second time before entering the appropriate instructions. Note that you will be using only the middle switches to enter instructions and addresses, since the addresses do not get very large.

5) To speed things up, simply enter successive instructions into the data switches and flip the DEPOSIT-NEXT switch down after entering each one; this procedure automatically stores the instructions into consecutive memory locations.

6) When you are done entering the program, enter address 0 again, and flip the EXAMINE switch up. Check to see if the data bus lights indicate the correct machine language instruction. If they do not, enter the correct instruction on the data switches and press DEPOSIT. Repeat this step if you had to make a change, otherwise flip the switch to EXAMINE-NEXT (down), and see if address 1 is okay. Address 1 should be displayed on the address lights, and 11111111 should appear on the data bus lights (i.e. all the lights should be on).

7) EXAMINE-NEXT your way through the rest of the program, and change anything that is wrong as you go. When you think everything is okay, set the address to 0 again and press EXAMINE.

8) Now flip the RUN switch up and see what happens.

9) To test the program try flipping some of the switches labelled address programmed input and see if the lights corresponding to them light up on the output lights. If a switch is in the up position, a light should be on, and if a switch is in the down position, the light should be off.

10) Now flip the STOP switch down, and then flip the RESET switch up.

11) If your program is not working, go back to step 6; otherwise, try using the SINGLE STEP switch by flipping it up or down and seeing what happens. Flip the switch several times. What you are doing is executing your program one machine cycle at a time (a machine cycle is part of a machine language instruction). You can now see the way in which the address lights, data lights, and output lights change as each instruction is executed. You may also notice strange things appearing on the output lights at certain times which are the result of machine cycles you didn't actually program in, but which the computer performs automatically. All of this is displayed when you run a program, but it is displayed and changed so fast that you can not see it. By SINGLE STEPPING you can view the machine operation at your leisure.

12) When you are done playing around, turn OFF the power, please. (PWR OFF switch flipped down)

HAVE FUN !!!!!!!!!

Focus on the Assembly Language for Our Microcomputer

Basically our microcomputer can be programmed to do only five different kinds of things. (For further clarifications of the ideas presented here, see Chapter

4 of the Intel 8080 Microcomputer System User's Manual).

1) Perform operations that require moving information around between different parts of the system.

2) Perform arithmetic operations on specified data.

3) Perform logical operations on specified data.

4) Resequence the normal (sequential) flow of instructions.

5) Monitor and control the internal operations of the machine.

(If you are unfamiliar with the basics of computer software, you may find it beneficial to read the section entitled Fundamental Software Concepts at this time).

1) Instructions of this type belong to what is called the data transfer group. These instructions allow data to be moved around between registers or between memory and the registers. These kinds of instructions are necessary to allow data to be moved into the registers where it can be manipulated (by other types of instructions) and later placed back into memory.

2) Instructions of this type belong to the arithmetic group. These instructions allow data in the registers and memory to be added, subtracted, incremented by 1, or decremented by 1. They form the heart of most programs performing any sort of arithmetic operations. The programmer must figure out ways to perform multiplications, divisions, and other more advanced mathematical operations and functions by using combinations of the above instruction.

3) Instructions of this type belong to the logical group. These instructions are used to manipulate and compare all of the bits in a data word stored in memory or in a register with another data word stored in memory or in a register. The bits can be ANDed, ORed, EXCLUSIVE - ORed (abbreviated XOR), compared, rotated, or complemented. These instructions are used to perform comparisons and change data values by non-arithmetic operations.

A data word consists of 8 bits, each bit being one binary digit, either a 0 or a 1,(e.g. a data word may look like 10111001).

Complementing a data word involves changing 0's to 1's and 1's to 0's. The complement of the above data word is 01000110.

Rotating a data word involves shifting the bits either to the right or to the left, with the bit "falling off the end" being placed on the other end. For example, using 10111001 and rotating left produces 01110011, while rotating the original word right produces 11011100.

Comparing involves subtracting the contents of a register or memory location from the accumulator and setting the appropriate condition flags (explained under group 4 instructions). These instructions are usually used before performing tests that will determine conditional transfers of control based on the status of the condition flags, since the instructions set the status of those flags.

ANDing two words involves comparing the corresponding bits of the two words to produce a resulting word based on the following rules. 1 AND 1 is 1, 1 AND 0 is 0, 0 and 1 is 0, 0 AND 0 is 0. Example: 10111001
01100011
produce 00100001 when ANDed

ORing is similar to ANDing except that the following rules apply: 1 OR 1 is 1, 1 OR 0 is 1, 0 OR 1 is 1, 0 OR 0 is 0.

Example: 10111001
01100011
11111011

XORing is also similar to the above operations, but with the rules: 1 XOR 1 is 0, 1 XOR 0 is 1, 0 XOR 1 is 1, 0 XOR 0 is 0.

Example: 10111001
01100011
11011010

4) Instructions of this type belong to the branch group. Before discussing the instructions, it is useful to know about the condition flags which exist in the system. There are five such flags associated with the execution of each instruction; some instructions affect some or all of the flags, while others leave them unchanged. A flag is just a bit which is given a name, and it takes on the value 1 when set,

0 when reset. The 5 flag bits are stored in a special register in the computer and do not belong to any particular data word. The flags are set to 1 under the following conditions:

Zero- if the result of an instruction has the value 0.
Sign- if the result of an instruction has a 1 in the leftmost bit.
Parity- if there are an even number of 1's in the resulting word.
Carry- if the instruction resulted in a carry from addition or a borrow from subtracting or comparison in the leftmost bit.
Auxiliary carry - if there is a carry out of the 4th bit from the right into the 5th bit from the right of the resulting value (generally used when working with decimal number equivalents stored in binary form).

The branch group instructions can examine the condition flags, and they can alter the sequence in which instructions are performed by specifying where to look for the next instruction, the specification being based upon the status of the condition flags. Other instructions within the group are "unconditional" instructions, which alter the sequential flow of instructions without testing the condition flags. Instructions which issue subroutine calls and perform returns from subroutines are also a part of this group. These operations may also be performed either conditionally or unconditionally.

5) The instructions in this group belong to the machine control group. The group includes instructions which reference and maintain the stack as well as the I/O (input/output) devices. A stack refers to some portion of memory which is temporarily set aside to hold specific data values in a particular order. The last data value entered into the stack is the only one which can be retrieved immediately; once it is retrieved the value entered prior to that one can be recovered, and so on down the line. Data can be added to or removed from the top of the stack at any time. Instructions exist for putting data into the stack and for taking it out of the stack. Special instructions also allow the status of the processor (defined as the contents of the accumulator and the condition flags) to be placed onto and removed from the stack. In this way, the computer

can respond to an interrupt or process a subroutine call and then come back and pick up where it left off, since it can save all the information it needs for continuing.

Instructions exist for informing the machine that it should or shouldn't allow itself to be interrupted while performing some sequence of operations, or that it should stop doing everything (HALT), or that it should perform no operation for a while (sit idly until told it can go on - WAIT or HOLD).

Instructions also exist which inform the machine where it should get the next data word and where it should place the data it is done working with; these specifications might involve registers, memory locations, or particular I/O devices such as card readers, line printers, or disk drives.

Other Useful Things to Know About Assembly Language Programming

Even though each assembly instruction is represented by one word in the computer, some of the instructions require that additional information be supplied to the computer before they can be executed. For instance, an instruction which tells the computer to move data from a memory location to a register must tell the computer three things: First, that it is a move instruction; Second, the address of the data in memory; and Third, which register to move the data into. Because all of this information will not fit into 1 word of computer information (8 bits), several words must be used to store all of the information.

First, all instructions are stored in one word. When the computer receives an instruction it determines whether or not the instruction is the kind that requires more information. If it is, then the computer gets that information from a predefined place in memory or a register, in a fashion that is also predefined for each instruction. If no additional information is required, the computer executes whatever it is that the instruction tells it to do. This

gives rise to what is referred to as varying length instructions, since an instruction will be thought of as not only the instruction itself, but also all of the necessary address and register specifications which are required to execute that instruction. On our system, instructions can be one, two, or three words long, and if they are more than one word long, the individual words are stored sequentially in memory.

It is a peculiarity of our system (and of many other systems) that an address must be broken up into 2 words (since an address requires 16 bits to be specified, and a word contains only 8 bits). The rightmost 8 bits of the address (or least significant part) are stored in the first word following the instruction, and the leftmost 8 bits are stored in the 2nd word following the instruction. For example, if an address were 1011001100110011, then the first word following the instruction accessing this memory location would contain 00110011, and the word after that would contain 10110011.

There are four different ways in which the required data for an instruction can be specified. These ways are referred to as addressing modes, and they work in the following ways:

Direct - the most common way in which the address of the data word is specified directly after the instruction specification itself.

Register - the instruction specifies the register or register-pair in which the data is located. A register-pair is 2 registers which are arbitrarily assigned to work together in specifying addresses requiring 16 bits.

Register-indirect - the instruction tells the computer which register or register pair to look at in order to find the address of the word which actually contains the data. Hence the computer will have to perform several operations:

1) Get the address of the data from the register or register-pair.
2) Go to that address and use the information stored there as the data.

Immediate - the instruction actually doubles as both the instruction and all or part of the data itself.

Fundamental Software Concepts

Computers do not function without some human intervention. The way in which people interact with computers is through programs which tell the computer what it should do. Programs fall under the heading of computer software, which is that part of the computer not directly related to the physical parts out of which it is built (referred to as hardware). Software is a term used to refer to the progress and procedures available on a computer.

There are many levels at which computers can be told what to do, or programmed. The most common of these employs "higher level" languages such as BASIC, COBOL, or FORTRAN. The reason languages are called "higher level" is that they do not communicate directly with the computer; several stages of translation must be performed on programs written at the higher levels before they become understandable to the computer.

Basically, the computer is a most unsophisticated advice. All it can do is distinguish between something that is either happening or not happening, or cause some action either to take place or not to take place. It has become customary to represent the occurrence of an event in a computer by a 0 or a 1 (with the understanding that the lack of occurrance of an event assumes the value which the occurrence does not, e.g., if the occurrence of an event has value 0, the lack of occurrence of that event has value 1). Hence, a 0 might be used to indicate that no electricity is flowing in a wire (where a 1 would indicate that electricity is flowing in that wire), or a 0 might mean it is time to perform a special task (where a 1 would mean not to perform the task at that time).

In order to be able to talk to a computer, one must be able to tell it what things are or are not happening, and what things are or are not supposed to happen (in the order in which the computer expects to be told these things);

in so doing, one is giving instructions to the computer. The most basic means for this communication with a computer is through the specification of instructions using 0's and 1's in a specific order. If this is being done, one is said to be operating at the machine level, and programs written using only 0's and 1's are said to be written in machine code. Machine code is the only "language" a computer really "understands". Because humans find it extremely difficult to work at this level, easier means of communication with the computer have been developed. The basic idea behind all of them has been to allow people to write things down in a way which they can easily understand. Some previously written programs can also understand, translate, and send along the instructions to successively lower levels, until at last some stage produces the machine code equivalent to what was originally specified.

The stop right "above" the machine level is called the assembly level. The assembly level sets up a direct correspondence between every possible computer instruction allowed (in terms of 0's and 1's) and some meaningful mnemonicname for each instruction (which indicates what that instruction does). Hence, the person wishing to communicate with the computer (the programmer) can now write down what needs to be done using mnemonic names, called assembly language instructions, and thereby produce an assembly language program (also called a source program). This source program is then fed into an assembler which converts each assembly language instruction into its corresponding machine instruction, thus producing what is called the object code. (Note that the assembler is itself a program, written in machine language, and, the object code is the machine level equivalent of the assembly language program.)

The next level involves the "higher level" languages. These languages are composed of statements which do not necessarily have a direct correspondence to machine level instructions; instead, they correspond to frequently used sequences of machine level instructions. There are two ways in which higher level languages

are implemented on a computer. The first is the use of an interpreter program. The interpreter functions just like an assembler except that it accepts only one higher-level programming statement at a time, and immediately converts that statement into machine language, so that the machine language program is built up from one higher level language one instruction at a time. Diagnostic error messages are usually given to the programmer if an illegal or undecipherable higher level language statement is entered. Interpreter programs are themselves usually written in assembly or machine languages.

The other common way to implement higher level languages is through a compiler. A compiler is a program similar to an interpreter. It differs in that it accepts an entire program written in the higher level language at once. It then produces either the equivalent machine code for the entire program or an assembly language equivalent program which can be fed into an assembler to produce the equivalent machine code.

In general, for a given higher level language, an interpreter program will be shorter than a compiler, since it need recognize fewer things about the program. A compiler, on the other hand, produces more efficient machine code because it can take into account certain patterns that an interpreter will never notice.

In order to efficiently manage all of the resources available to a computer, sequence the execution of programs, and keep track of accounting records, most computer systems contain an operating system. The operating system is one or more programs which have been written specifically to perform such functions. It is really the operating system which controls all of the operations which the user sees occurring in the computer system.

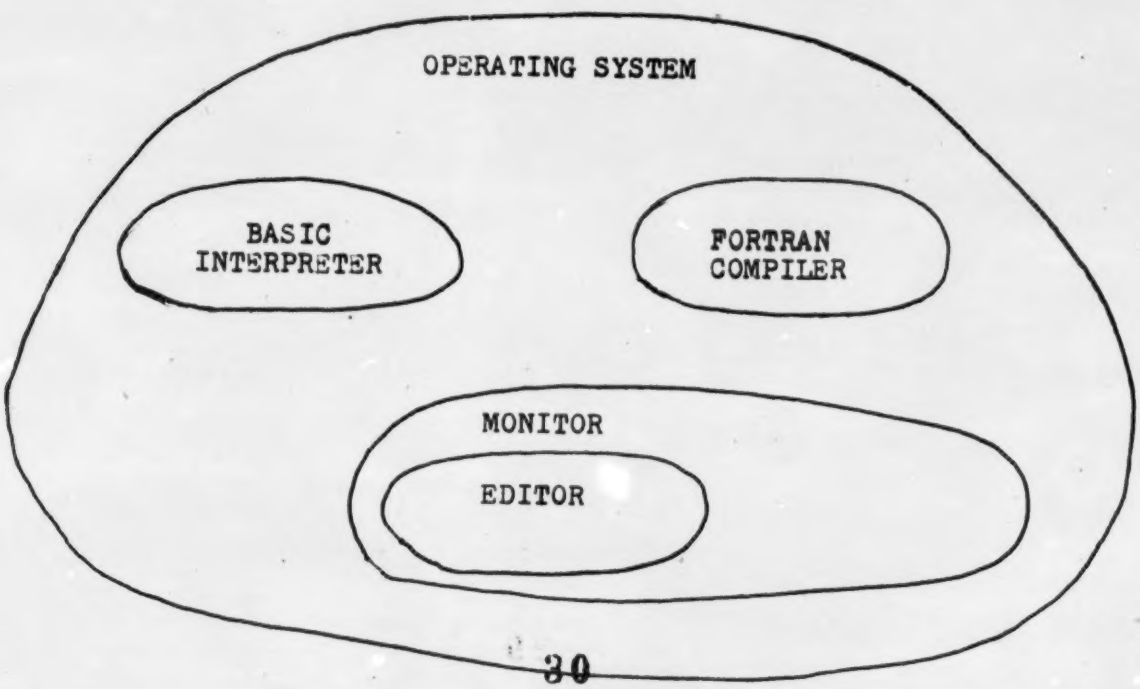
Often, operating systems are designed in modules. One module of an operating system is frequently a separate program called a monitor, which a user

can use to enter programs into the computer. When a user is allowed to assess a monitor program, he can usually obtain information about system data, records, and activity; he may also create, modify, or remove programs. The monitor is essentially a program which facilitates interaction between the user and the computer, but which is under the control of the operating system.

A monitor may also consist of modules. One of the most common parts of a monitor is called an editor. An editor is essentially a program which aids the user in altering or creating a program or data file.

The programmer must be able to distinguish between the commands which are instructions to various system programs--(such as the operating system, monitor, and editor)--and the programming statements (or instructions) which belong to the language in which programs are being written. Frequently, the commands which a programmer sees are intended for moving around between the various system programs. Commands are also used to obtain information about the status of various developments in the system or to specify that a particular action should take place.

DIAGRAM OF A POSSIBLE SYSTEM SOFTWARE STRUCTURE



References

IMSAI 8080 Microcomputer System User Manual. IMS Associates, Inc., 1976.

Osborne, Adam, An Introduction to Microcomputers. (Volume 1, "Basic Concepts," and Volume 2 "Some Real Products"), Adam Osborne and Associates, Inc., 1976.

8080 Microcomputer Systems User's Manual. Intel Corporation, September, 1976.

Almost all of the information in the present report and in ISEAC Number 26 (An Introduction to Microcomputers) come from the above sources. The author has interjected personal feelings or experiences wherever he felt they were appropriate or necessary to help clarify a difficult point.